Distributed FRANC Simulator

Extensions and Improvements

Aurélien Frossard (aurelien.frossard@epfl.ch)
Boris Danev (boris.danev@epfl.ch)

August 2004

Distributed Systems Laboratory (LSR)

David Cavin (david.cavin@epfl.ch), Supervisor
Yoav Sasson (yoav.sasson@epfl.ch), Advisor

Contents

1 New Features and Improvements

This section describes the issues which were addressed during the second phase of the project on the Distributed FRANC Simulator. It primarily addressed the following issues:

- Distribution of the information and simulation startup procedure
- Enhance the configuration utility and add new functionalities
- Revisit the logging procedure
- Create validation error limits for the simulation process
- Coverage analysis of the code
- Some performance evaluation

Issues from the previous release which were not addressed:

- Build a graphical user interface for the simulator. This is only a plus but not an important issue. That's why it was not addressed by the second phase.
- Interpretation of the log information. Currently the simulator logs each event which occurs during simulation. Log interpretation is completely independent of the current project.

1.1 Distribution and simulation startup

In the previous release, each node was started individually by the user on each computer taking part in the distributed simulation. Running many large scale tests was painful.

A lightweight server called daemon that has the ability to start nodes was developed to address this issue. The simulation is now launched from a single host and the coordinator distributes the load among different computers using the daemons. It tells each daemon to start some nodes with the provided configurations. Once started the node connects to the coordinator and simulation process continues normally.

The daemon was developed with a multi-session functionality. This feature allows the user to start many simulation processes using the same daemon. This could be very useful when trying to simulate different networks at the same time. When doing this attention should be payed on the configuration files of the different simulation network aka config files. The configurations should not contain the same

names for the log files, or the same network ports for communication. Otherwise two different simulations could interfere and provide erroneous results.

Each daemon logs its actions in log file called daemon.log. It could be consulted in case of problems during simulation.

1.2 Configuration utility

We addressed two important issues while revisiting the configuration utility of the simulator.

The first one was related to the internal checks about the validity of the simulation configuration file. We wanted to limit to minimum any possible faults made by the user when writing configuration files. The xml configuration file structure was described using a DTD (Document Type Definition), which is a rather outdated way of describing xml files. We now use today's standard XMLSchema. Besides being more trendy, this allowed us to add more type-checking in the document description itself.

The second one was to add the possibility to choose manually the mobility pattern a given node will follow. This feature was suggested by our supervisors.

1.3 Logging procedure

Log4J is very limited when trying to separate logging information by levels. Indeed it is possible, but only by manipulating Log4J XML configuration files which would have added a lot of complexity in the simulator for a very little result. That is why we abandonned the idea to provide other layers with possibility to use also Log4J for logging application specific information. If the user wants to log information proper to its own application, it should do it by its own means, using for example the default Java logging package or some other packages available in the community.

Another functionality of the simulator is that it keeps track of the standard output of each simulation node started. It could be consulted in files with extension "'.stdout"' for each node. However, we strongly recommend to never use the standard output for logging.

The event logging architecture was time and memory consuming in the previous release. Four events were very frequently used during a simulation: *NewLocationEvent*, *NewScheduleEvent*, *MessageReceivedEvent* and *MessageSentEvent* and each time one of these events occurred, a new object was created, logged and then destroyed. This issue was addressed using the singleton pattern.

1.4 Validation of the simulation process

The simulator now provides a way to validate the simulation process through detection of synchronization errors. We introduced two types of errors:

1. GLOBAL SYNCHRONIZATION ERROR might occur if there is a deviation between the real end simulation time mesured by the simulator and the one mesured by the simulation node. The absolute value of the difference between these two times should not exceed the simulation step (clock speed) defined in the config file. This kind of error may occur when the clock of the simulator's machine is not synchronized with the clocks of the other machines where simulation nodes were run.

2. LOCAL SYNCHRONIZATION ERROR concerns only the simulation node itself. It mesures the difference between the theoretical end time and the real end time of the simulation in the node. The absolute value of the difference should not exceed the simulation step (clock speed). The theoretical end time is provided by a Timer task using the theoretical values calculated at the beginning of the simulation and the real time is provided by the number of clock ticks that a simulation node has seen from its local clock during the simulation. If the absolute value exceeds the simulation step, it means that the local clock was too late and did not succeed to issue the estimated theoretical number of ticks to the simulation node. This is normally due to too many Java virtual machines running on the same machine.

If any of the errors mentionned above occurs, they are logged in the log file of the concerned node. That is why, it is absolutely necessary after simulation to check each end of file for this kind of errors. The errors are signaled by the event called "'**SimulationWarningEvent**"' logged at the end of the log file.

If NTP protocol runs in the simulation cluster, the GLOBAL SYNCHRONIZATION ERROR should never occur.

If the user tries to execute too many simulation nodes on the same machine, this could lead to overloading the machine and provoking a LOCAL SYNCHRONIZATION ERROR. If this error occurs, it is important to look for the exact value of the difference which is logged at the end of the log file. If this value exceeds with very little the simulation step, it means that the machine is on the limit of overloading and the user can still accept the results if the simulation was run long enough. Please refer to the next chapter for some practical performance issues.

2 Performance

2.1 Coverage Analysis

During the second phase of the project, we performed a coverage analysis of the code in order to achieve 90 percent level of coverage of critical classes of the application such as the SimulationLayer, ManetSimulator, SimController and SimConfigurator classes. Most of the time it was really hard to increase the coverage percentage, because it involved provoking quite complicated crash scenarios which never occurred during the testing period.
We used Clover[15] for a coverage analysis tool and we found it extremely well done and powerfull to track code which was not executed during testing. Unfortunately Clover is not free which forced us to use only the evaluation version.

2.2 Performance evaluation

While testing the code for correctness, we performed some performance evaluation tests in order to get a rough idea of the stability of the simulator and estimate the number of simulation nodes per machine that can be run. In the LSR cluster we used the two different types of machines available:

1. Processor: Intel(R) Pentium(R) 4 CPU 1.50 Ghz Stepping: 7 Cache size: 256 KB RAM Memory: 256 MB

2. Processor: Intel(R) Pentium(R) 4 CPU 1.80 Ghz Stepping: 4 Cache size: 512 KB RAM Memory: 512 MB

The base for the evaluation was a 12 hour simulation of a network of identical simulation nodes based on the ch.epfl.lsr.adhoc.simulator.testing.SimpleTestLayer. Each node was broadcasting messages every 2 seconds. We mesured the number of nodes a machine of type 1 and 2 were able to simulate during 12 hour period respecting the two types of error limits defined in Section 1 Validation of the simulation. Each machine was never charged with additional processes. The only processes running were the standard root processes and the simulation processes. The main simulator was run on a different machine. After two weeks of mesurements of different configurations, we came to the following conclusion:

- Machine of type 1 respected the error limits until it reached 6 nodes. When run with 7 and 8 nodes, the error limits were slightly exceeded. In this case it is up to the user to accept or not the simulation results. Remember that the longer the simulation is, the more unsignificant the error is.

- Machine of type 2 respected the error limit until it reached 10 nodes. We did not test more then 10 nodes, because we were satisfied with this performance which allowed testing large number of nodes on multiple machines.
- These tests allowed us to make the assumption that the available RAM memory on a machine is much more important than the CPU power. Indeed machine 1 started swapping when run with 7 or 8 nodes.
- We did not test the previous assumption, because this was beyond the scope of the project.

N.B! This section is far from being a full scaled performance evaluation of the simulator. The performance evaluation is by itself a semester project and we do not think it is needed to successfully run simulations.

3 Simulation guidelines

This section is intended to be a short manual for users of the simulator. It is not a developer's manual. It is similar to section 6 of [2] but includes the new features of the simulator.

3.1 Configuration

3.1.1 FRANC

There's nothing particular to do for the simulator here. Just configure a FRANC node as described in [3]. We will refer to the corresponding configuration file as `franc.xml`. You may wish to create many different configuration files for simulating a network containing different types of nodes.

3.1.2 Mobility

Mobility patterns must follow the format used for the NS2 simulator[1]. Such patterns can be written manually or generated automatically using the setdest[2] utility provided with NS2. Rather than giving the complete grammar of NS2's mobility patterns we prefer giving an explained example of the format accepted by our parser.

First of all a mobility file can contain as many comments as ones wishes. Of course, comments are ignored by our parser, as well as empty lines.

```
#A comment starts with # and stops at the end of the line
```

The file must begin with node definitions. A node definition defines the node's initial position along X, Y and Z axes, but the value on Z is always null. Three lines must be used, one for each axe. Here is an example of a file defining three nodes.
Note that the node IDs are in increasing order starting from zero.
One cannot define a node ID equal to one, if there is no node with ID equal to zero.

```
$node_(0) set X_ 380.825962687796
$node_(0) set Y_ 376.809106063166
$node_(0) set Z_ 0.000000000000

$node_(1) set X_ 961.976277430324
```

[1] see [4]
[2] in NS2's distribution 2.26, it is located under indep-utils/cmu-scen-gen/setdest

```
$node_(1) set Y_ 842.259984507709
$node_(1) set Z_ 0.000000000000

$node_(2) set X_ 781.539181377774
$node_(2) set Y_ 715.421886604222
$node_(2) set Z_ 0.000000000000
```

Mobility patterns follow, as an ordered sequence of orders.

```
$ns_ at 2.0000 "$node_(0) setdest 376.8583 552.9234 7.9263"
$ns_ at 2.0000 "$node_(1) setdest 135.0701 119.8773 7.1105"
$ns_ at 2.0000 "$node_(2) setdest 48.5678 128.8905 4.4397"
$ns_ at 24.2244 "$node_(0) setdest 376.8583 552.9234 0.0"
$ns_ at 26.2244 "$node_(0) setdest 465.0890 118.74282 0.0670"
$ns_ at 156.41959 "$node_(1) setdest 135.07012 119.8773 0.0"
$ns_ at 158.41959 "$node_(1) setdest 542.8177 303.2304 6.02286"
$ns_ at 213.44505 "$node_(2) setdest 48.5678 128.8905 0.0"
$ns_ at 215.44505 "$node_(2) setdest 129.0663 136.3148 9.6791"
```

Each line represents an order for a node. The first number represents the time at which the new order occurs. The number following `$node` naturally indicates the concerned node and the last three numbers indicate the new destination (as a two dimensional point) and the new speed. The lines are sorted by increasing order of their time field.

When generating a mobility file with NS2's setdest utility, a special node `god` may appear:

```
$god_ set-dist 1 2 1
$ns_ at 47.3787 "$god_ set-dist 0 1 2"
```

Such lines are ignored by our parser.

Important notice when writing a mobility file by hand

Pauses must be explicit: when a node reaches its destination, it must have an order that tells him to go elsewhere or to stay at his current location. Due to the structure of NS2's mobility patterns, it is hard for the parser to know for certain if there is an implicit pause or not. NS2's *setdest* only generates explicit pauses. You can find below the pattern for writing pauses. The writer of the file must compute himself the value of `someTimeT2`: `someTimeT2` = time value when the node reaches its destination (`someX`,`someY`).

```
$ns_ at someTimeT1 "$node_(0) setdest someX someY someSpeed"
...
$ns_ at someTimeT2 "$node_(0) setdest someX someY 0.0"
```

3.1.3 Simulator

Outline

The simulator's XML configuration file is divided in sections, each of them concerning a different group of options: *SimulationConfig*, *SimulationDaemons*, *SimulationLayer* and *SimulationNetwork*, which itself contains the two subsections *global-config* and *custom-config*. Here is an outline of the file:

```
<Simulator
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="sim-config.xsd">

  <SimulationConfig>  ...   </SimulationConfig>

  <SimulationDaemons> ...   </SimulationDaemons>

  <SimulationLayer>   ...   </SimulationLayer>

  <SimulationNetwork ...>
    <global-config> ... </global-config>

    <custom-config>
      <node id="1" ...> ... </node>
      <node id="2" ...> ... </node>
      <node id="3" ...> ... </node>
    </custom-config>
  </SimulationNetwork>

</Simulator>
```

The first two lines after `<Simulator` indicate that the file must be validated against the XMLSchema `sim-config.xsd`. This schema replaces the old DTD and allows for a much stronger type checking.

SimulationConfig

It contains some global parameters for the simulation and its distributedness. The first parameters concerns the server component of the main coordinator. These parameters allow to personalize the server TCP connection socket options. The ip and port fields indicate the port and ip of the listening server, the timeout specifies the connection timeout in milliseconds for each client trying to connect to the server and queue field indicates the number pending connections the server should queue. The clock parameter indicates the clock speed of the simulation process

also in milliseconds. The duration field indicates the end time of simulation in milliseconds and the output-dir field gives the possibility to customize the output directory for storing all files generated by the simulator.

Example:

```
<SimulationConfig>
  <server ip="127.0.0.1" port="7666" timeout="20000" queue="20"/>
  <clock>1000</clock>
  <duration>30000</duration>
  <output-dir>logs</output-dir>
</SimulationConfig>
```

SimulationDaemons

This is a list of the daemons available for the simulation. Each daemon will host one or many frameworks during the simulation : the simulator will distribute the different frameworks it has to run among the daemons listed here. Each deamon is identified by a hostname or ip adress and a port number.

Example:

```
<SimulationDaemons>
  <daemon ip="123.456.789.10" port="8888"/>
  <daemon ip="123.456.789.11" port="9999"/>
  <daemon ip="123.456.789.12" port="9999"/>
</SimulationDaemons>
```

SimulationLayer

This section only specifies the SimulationLayer Java class. It is used to replace the data-link layer of a FRANC node. The corresponding *DataLinkLayer* parameters in `franc.xml` are replaced by the ones provided here. Unless multiple implementations for this class are provided, there's nothing worth modifying here.

Example:

```
<SimulationLayer>
  <name>SimulationLayer</name>
  <class>ch.epfl.lsr.adhoc.simulator.layer.SimulationLayer</class>
</SimulationLayer>
```

SimulationNetwork

The size of the network should be indicated in the attribute *size*. The second attribute allows you to specify the mode to be used by the simulator:

auto In this mode, you do not need to specify the network topology in *custom-config*. The simulator uses only the information defined in the *global-config*. This means that it will create a default network with *size* nodes and each node will inherit the configuration by the default specified in *global-config*.

manual In this mode, the user should specify the *custom-config* element and for each node which should be different from a default node, specify the particular behavior.

Example:

```
<SimulationNetwork size="3" mode="manual">
  <global-config> ... </global-config>

  <custom-config>
    <node id="1" ...> ... </node>
    <node id="2" ...> ... </node>
    <node id="3" ...> ... </node>
  </custom-config>
</SimulationNetwork>
```

global-config

This field allows the user to provide default information for the network nodes:

node-ConfigIn User provides the default configuration file for a FRANC node according to the specification of FRANC.

node-ConfigOut The simulator modifies the file specified in the node-configIn field and creates a new file with the name provided in this field. Actually, this is the file used to start the node in simulation mode

node-errorLog User provides the name of the file that should be used to log error messages for a simulation node.

node-simulationLog User provides the name of the file which should be used to store simulation relevant information. Actually, this is the file to consider during the data processing of a given simulation process.

node-transmissionDefaults User should provide the default transmission range and transmission quality of a simulation node. The unit of the range could be anything, but it should be interpreted according to the units in the mobility pattern. If the speed of a node is in meters per second, the range should be considered in meters. The transmisssion quality must be a number between 0 and 1, 1 meaning full quality. This number is interpreted as the probability of not losing messages.

mobility-pattern User provides the mobility file (generated by NS2 tool setdest) and specifies the time scale to be used when interpreting the mobility pattern. This is needed because *setdest* specifies no units. The time scale has to be expressed in milliseconds. When time values are read from the mobility file, they are multiplied by the time scale. Thus a value of 1000 should be used when dealing with a mobility file expressed in seconds.

Example:

```
<global-config>
  <node-configIn>franc.xml</node-configIn>
  <node-configOut>franc_ready.xml</node-configOut>
  <node-errorLog>error.log</node-errorLog>
  <node-simulationLog>simulation.log</node-simulationLog>
  <node-transmissionDefaults range="100" quality="1"/>
  <mobility-pattern timeUnit="1000">mobility.txt</mobility-pattern>
</global-config>
```

custom-config

Thanks to this element the user could override the default behavior specified in the *global-config*. For most of the elements and attributes, if some of them are not specified, they take automatically the default values of *global-config*. In the custom-config, the user could also specify an on/off schedule for a given node. The schedule unit is milliseconds and the attribute *scaleBy* saves you time when writing the milliseconds.

The *node-mobilityID* element allows the user to chose which of the n mobility patterns should be used for the node. Here are some explainations: the mobility file contains n mobility patterns from id 0 to $n-1$. Since a node with id 0 is not allowed by the framework (it means broadcast), the default mapping is to give node i the mobility pattern $i-1$. For some reason the user may want to have more control over this mapping, which is why the *node-mobilityID* element exists. It allows the user to specify a manual mapping between nodes and mobility patterns of the mobility file. In the following example we gave node 3 the mobility pattern 1. As you can see, we also specified a *node-mobilityID* for the other nodes. This is

because partial manual mappings is not recommended: it could lead to an undesired situation where two nodes would have the same mobility pattern. Our advice is either use the default (by specifying nothing), or specify the *node-mobilityID* element for *all* nodes.

Example:

```
<custom-config>
  <node id="1" range="50" quality="0.5">
    <node-configIn>config/nodes/in/simpleFranc.xml</node-configIn>
    <node-configOut>config/nodes/out/00.xml</node-configOut>
    <node-schedule scaleBy="1000"> - scale by 1000 to get seconds
      <on atTime="0"/>   - at time 0 seconds
      <off atTime="10"/> - at time 10 seconds
      <on atTime="20"/>  - at time 20 seconds
    </node-schedule>
    <node-mobilityID>0</node-mobilityID>
  </node>

  <node id="2">
    <node-mobilityID>2</node-mobilityID>
  </node>

  <node id="3" range="120">
    <node-mobilityID>1</node-mobilityID>
  </node>

</custom-config>
```

3.1.4 Log4j

The simulator provides two types of logging. The first one is only for the main simulator and can be freely configured through an XML file which has to be provided as a command line argument. This type of logging is used only inside the coordinator component of the simulator in order to log any error messages during the whole simulation process. Remember that the simulation process is supervised by the coordinator. The user is not supposed to modify this file, thus no documentation is provided here.[3] Just use the provided file. The second type of logging is inherent to the simulator and is stored in a special Java class. It should never be changed without carefully exploring the logging process of the simulator.

[3] if you are interested in Log4j, please refer to [8] and [9]

3.1.5 Use cases

For those who did not bother reading the previous pages, here is a brief summary of what to do for running a simulation. If at any point you are lost, it is probably time for you to consider reading section 3 from the beginning, that is from page 7.

Only one type of node

Fairly easy! Let `myFranc.xml` be the configuration file of the nodes you want to simulate, let `myMobility.txt` be the mobility file you want to use, let N be the number of nodes you want to simulate, let RANGE be their transmission range and let T be the duration of the simulation (in milliseconds). Suppose that you have three daemons running on three computers and listening on port 8888. Edit the simulator's configuration file as follows (only modified parameters are shown):

```
<Simulator>
  <SimulationConfig>
    <server ip="YOUR_IP_GOES_HERE" ... />
    <duration>T_VALUE_GOES_HERE</duration>
  </SimulationConfig>

  <SimulationDaemons>
    <daemon ip="123.456.789.10" port="8888"/>
    <daemon ip="123.456.789.11" port="8888"/>
    <daemon ip="123.456.789.12" port="8888"/>
  </SimulationDaemons>

  <SimulationNetwork size="N_VALUE_GOES_HERE" mode="auto">
  <global-config>
      <node-configIn>myFranc.xml</node-configIn>
      <node-configOut>myFranc_ready.xml</node-configOut>
      <node-errorLog>error.log</node-errorLog>
      <node-simulationLog>simulation.log</node-simulationLog>
      <node-transmissionDefaults
                  range="RANGE_VALUE_GOES_HERE" quality="1"/>
      <mobility-pattern timeUnit="1000">
                  myMobility.txt
      </mobility-pattern>
    </global-config>
  </SimulationNetwork>
</Simulator>
```

This configuration of simulation process will create a network of N nodes obeying the mobility pattern `myMobility.txt`. Each one of them uses default FRANC configuration `myFranc.xml`, logs error events in `error.log` and simulation events in `simulation.log`. They have all a default range of transmission RANGE and perfect quality of 1.

Multiple types of node

Not that difficult! Begin by doing the same as described above, then continue editing the configuration file with some more assumptions: all nodes are configured using the default values except nodes 23, 45 and 12 which have their own configuration files, transmission ranges and qualities. For example let's say that node 23 has a transmission range of 432, a transmission quality of 50%, uses mySpecialFranc.xml instead of the default myFranc.xml and is only active for 3 minutes. Here's what the configuration file would look like:

```
<SimulationNetwork ... mode="manual">
  <global-config> ... </global-config>

  <custom-config>
  <node id="23" range="432" quality="0.5">
    <node-configIn>mySpecialFranc.xml</node-configIn>
    <node-configOut>mySpecialFranc_ready.xml</node-configOut>
    <node-schedule scaleBy="1000">
      <on atTime="0"/>
      <off atTime="180"/>
    </node-schedule>
  </node>

  <node id="45" ...> ... </node>
  <node id="12" ...> ... </node>
</custom-config>
```

3.2 Logging

A simulation node logs two types of events: error and simulation events. The user could provide the storage files for these events in the *global-config* using fields *node-errorLog* and *node-simulationLog*. Simulation events are logged in a unified manner using XML. A log entry begins with a header that contains the class name of the event and the simulation and system time at which the event occurred. The header is followed by the parameters specific to the event. Here is an example of a log entry:

```
<SimulationEvent
  name="ch.epfl.lsr.adhoc.simulator.events.NewLocationEvent"
  simTime="0"
  systemTime="Tue Feb 03 16:51:03 CET 2004">

    <param name=x>100.0</param>
    <param name=y>60.0</param>
</SimulationEvent>
```

3.3 Simulation startup procedure

In order to start a simulation process based on a network configuration file described in the previous sections, the user should refer to the Quick Start Guide available in the README provided with the distribution of the simulator. The guide describes an automated way to manage remote daemons and start simulations.

Here we simply provide the not automated way to start a remote daemon and the simulator. This may be useful when developing your own scripts for simulation.

The JAR file *MANETSimulation.jar* contains all the classes needed for running FRANC in simulation mode. You need this file to start a simulation process. The file is generated by the following ANT task available in the distribution:

$ant simulation

3.3.1 Remote Server (daemon)

The daemon can be started manually by using the following command in a folder containing the *MANETSimulation.jar* file on a machine with hostname *lsepc6.epfl.ch*:

java -cp MANETSimulation.jar ch.epfl.lsr.adhoc.simulator.daemon.RemoteServer -host lsepc6.epfl.ch -port 9999 -execjar MANETSimulation.jar -javapath /bin/java

Description of the options

-host	The user should provide the real hostname of the machine on the network
-port	The listening port of the daemon.
-execjar	The executable Jar file that the daemon should use to start simulation nodes
-javapath	The exact path to java is also needed

N.B! All options are compulsory and should contain correct values. Otherwise the daemon might not behave properly.

3.3.2 Main simulator

The simulator can be started manually by using the following command in a folder containing the *MANETSimulation.jar*, the myNetwork.xml and log4j-config.xml files:

java -cp MANETSimulation.jar ch.epfl.lsr.adhoc.simulator.ManetSimulator -config myNetwork.xml -log4j log4j-config.xml

Description of the options

-config	The simulation configuration file. See Section 3.1
-log4j	Log4J XML configuration file. Use the default one

References

[1] FRANC: A lightweight Java Framework for Wireless Multihop Communication
AUTHOR: *David Cavin, Yoav Sasson, André Schiper*
http://lsrwww.epfl.ch/ip9

[2] Distributed FRANC Simulator
AUTHOR: *Boris Danev, Aurélien Frossard*
http://lsrwww.epfl.ch/ip9

[3] Semester project: MANET Framework
AUTHOR: *Javier Bonny, Urs Hunkeler*
http://lsrwww.epfl.ch/ip9

[4] The Network Simulator NS-2
http://www.isi.edu/nsnam/ns/

[5] JEmu: A Real-Time Emulation System for Mobile Ad-Hoc Networks
AUTHOR: *Juan Flynn, Hitesh Tewari, DOnald O'Manhony*
http://www.cs.tcd.ie/omahony/jemu-iei.pdf

[6] Effective Java, Programming Language Guide
AUTHOR: *Joshua Bloch*

[7] Java Performance Tuning, Second edition
AUTHOR: *Jack Shirazi*

[8] The Log4j project
http://logging.apache.org/log4j/docs/

[9] The complete manual: Log4j
AUTHOR: *Ceki Gülcü*

[10] JDOM : a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code
http://www.jdom.org/

[11] Eclipse IDE
http://www.eclipse.org/

[12] Network Time Protocol (ntp)
http://www.ietf.org/rfc/rfc0958.txt

[13] Java Code Conventions for use in EDH
http://ais.cern.ch/apps/edh/CodingStandards

[14] The Not So Short Introduction to LaTeX 2_ε
AUTHOR: *Tobias Oetiker*

[15] Clover Coverage Analysis Tool
http://www.cenqua.com/clover/